НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ

“___” червня 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системне програмування»

зі спеціальності **123 «Комп'ютерна інженерія»**

на тему: Онлайн-гра клієнт-серверної архітектури

Виконала: студентка IV курсу, групи КВ-62
(шифр групи)

Баглай Іванна Юріївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доц. каф. СПіСКС к.т.н., доцент Орлова М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) __ (підпис)

Рецензент_____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» _____20___ р.

ЗАВДАННЯ
на дипломний проєкт студента

Баглай Іванна Юріївна

1. Тема проєкту «Онлайн-гра клієнт-серверної архітектури», керівник проєкту доц. каф. СПіСКС к.т.н., доцент Орлова М.М.,

затверджені наказом по університету від «__» _____ 20__ р. №_____

2. Термін подання студентом проєкту 22.05.2020

3. Вихідні дані до проєкту Назва. Онлайн гра клієнт-серверної архітектури
4. Зміст пояснювальної записки
   1. Аналіз існуючих рішень розробки онлайн-гри
2. Аналіз мережевих архітектур
3. Розробка онлайн-гри клієнт-серверної архітектури
4. Опис розроблених алгоритмів та підпрограм
5. Безпека передачі інформації
6. Оптимізація розробленої програми
7. Тестування програми

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) презентація; структурна схема топології клієнт-сервер з розробленою системою синхронізації; структурна схема архітектури програми; блок-схема алгоритму синхронізації кольору; блок-схема алгоритму синхронізації часу.

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	Ярослав КЛЯТЧЕНКО		

7. Дата видачі завдання 17.09.2019

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проекту	10.02.2020	
2.	Розроблення та узгодження технічного завдання	13.03.2020	
3.	Аналіз існуючих рішень	18.03.2020	
4.	Підготовка матеріалів першого розділу дипломного проекту	10.04.2020	
5.	Підготовка матеріалів другого розділу дипломного проекту	19.04.2020	
6.	Підготовка графічної частини дипломного проекту	1.05.2020	
7.	Оформлення документації дипломного проекту	10.05.2020	
8.	Попередній огляд матеріалів диплому на кафедрі	20.05.2020	

Студент _____ _Іванна БАГЛАЙ___

Керівник проєкту _____ _Марія ОРЛОВА___

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (58 с., 21 рис., 3 табл., список використаної літератури з 18 найменувань, 3 додатки).

Метою бакалаврського дипломного проєкту є дослідження та структуризація теоретичних відомостей, необхідних для розробки та реалізації онлайн-гри, і на основі отриманих даних розробити власну онлайн-гру клієнт-серверної архітектури.

Для досягнення поставленої мети проведено аналіз існуючих топологій з'єднання вузлів у мережі, шаблони проектів, призначених для синхронізації процесів, алгоритмів оптимізації кількості синхронізуючої інформації.

В результаті роботи було програмно реалізовано онлайн-гру, в якій було використано оптимальні та спеціалізовані під певну задачу алгоритми. Перевагою даного проєкту є абстрактність розробленої архітектури, що реалізує синхронізацію.

Результати дипломної роботи можуть бути використанні для вивчення основних концепцій онлайн-гри. А розроблена архітектура синхронізації може бути використана для розробки мережевих ігор.

При розробці онлайн-гри було використано ресурси: мова програмування C++, програма імітації умов мережі clumsy, програма візуалізації роботи потоків в реальному часі Tracy Profiler.

Ключові слова: онлайн-гра, клієнт-серверна топологія, реплікація об'єктів, навігаційна система обчислення шляху, затримка, транспортні протоколи, синхронізація.

SUMMARY

Qualification work includes an explanatory note (56 pages, 21 figures, 3 tables, list of references from 18 items, 3 appendices).

The purpose of the bachelor's thesis project is to study and structure the theoretical information needed to develop and implement an online game, and on the basis of the data to develop their own online game client-server architecture. To achieve this goal, the analysis of existing topologies of connection of nodes in the network, project templates designed to synchronize processes, algorithms for optimizing the amount of synchronizing information.

As a result of the work, an online game was implemented in software, in which optimal and specialized algorithms for a certain task were used. The advantage of this project is the abstractness of the developed architecture that implements synchronization.

The results of the thesis can be used to study the basic concepts of online gaming. And the developed synchronization architecture can be used for development of network games.

Resources were used in the development of the online game: C ++ programming language, a program for simulating the conditions of the clumsy network, a program for visualizing the work of streams in real time Tracy Profiler.

Keywords: online game, client-server topology, object replication, dead reckoning, path delay, transport protocols, synchronization.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ДП 045490.002 ТЗ	Онлайн-гра	4		
			клієнт-серверної			
			архітектури			
			Технічне завдання			
	А4	ДП 045490.003 ТП	Онлайн-гра	2		
			клієнт-серверної			
			архітектури			
			Відомість технічного			
			проекту			
	А4	ДП 045490.00.004 ПЗ	Онлайн-гра	59		
			клієнт-серверної			
			архітектури			
			Пояснювальна записка			
	А4	ДП 045490.01.005 Д1	Топологія клієнт-сервер	1		
			з розробленою системою			
			синхронізації.			
			Структурна схема.			

Змін.	Арк.	№ докум.	Підпис	Дата	**ДП 045490.001 ОА**		
Розробив		Баглай І.Ю.			Онлайн-гра клієнт-серверної архітектури **Опис альбому**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Орлова М.М					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-62	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ДП 045490.02.006 Д2	Синхронізація руху гравця	1		
			за допомогою			
			dead reckoning			
			Структурна схема.			
	А4	ДП 045490.03.007 Д3	Синхронізація кольору.	1		
			Блок-схема алгоритму.			
	А4	ДП 045490.04.008 Д4	Синхронізація часу.	1		
			Блок-схема алгоритму.			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічні матеріали.			

					ДП 045490.001 ОА	Арк.
						2
Змін.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Онлайн-гра клієнт-серверної архітектури».
Галузь застосування: ігрова індустрія, сфера розваг.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення онлайн-гри клієнт-серверної архітектури.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- сумісність з операційною системою Windows;
- можливість управління об'єктом;
- можливість синхронізувати обрайний у меню колір;
- можливість синхронізувати події у грі;
- можливість отримання статистики гравців;
- наявність зручного меню;

5.2. **Вимоги до апаратного забезпечення**

- Процесор: Intel Core i5-6200;
- Оперативна пам’ять: 8 Гб;
- Наявність доступу до мережі Internet (GPRS, EDGE, 3G, 4G);

5.3. **Вимоги до програмного та апаратного забезпечення користувача**

- Операційна система Windows;

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проекту	10.02.2020	
2.	Розроблення та узгодження технічного завдання	13.03.2020	
3.	Аналіз існуючих рішень	18.03.2020	
4.	Підготовка матеріалів першого розділу дипломного проекту	10.04.2020	
5.	Підготовка матеріалів другого розділу дипломного проекту	19.04.2020	
6.	Підготовка графічної частини дипломного проекту	1.05.2020	
7.	Оформлення документації дипломного проекту	10.05.2020	
8.	Попередній огляд матеріалів диплому на кафедрі	20.05.2020	

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ДП 045490.00.004 ПЗ	Онлайн-гра архітектури	58		
			клієнт-серверної			
			архітектури			
			Пояснювальна записка			
	А4	ДП 045490.01.005 Д1	Топологія клієнт-сервер	1		
			з розробленою системою			
			синхронізації.			
			Структурна схема.			
	А4	ДП 045490.02.006 Д2	Синхронізація руху гравця	1		
			за допомогою			
			dead reckoning			
			Структурна схема.			
	А4	ДП 045490.03.007 Д3	Синхронізація кольору.	1		
			Блок-схема алгоритму.			
	А4	ДП 045490.04.008 Д4	Синхронізація часу.	1		
			Блок-схема алгоритму.			

Змін.	Арк.	№ докум.	Підпис	Дата	**ДП 045490.003 ТП**		
Розробив		Баглай І.Ю.			Онлайн-гра клієнт-серверної архітектури **Відомість технічного проекту**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Орлова М.М					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-62	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічні матеріали.			

					ДП 045490.003 ТП	Арк.
						2
Змін.	Арк.	№ докум.	Підпис	Дата		

Пояснювальна записка
до дипломного проекту

на тему: Онлайн-гра клієнт-серверної архітектури

Київ – 2020 року

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

AI – Artificial Inteligence

CS – Client-Server

MTU – Maximum Transmission Unit

NAT – Network address translation

P2P – Peer-To-Peer

PCI DSS – Payment Card Industry Data Security Standard

PDU – Protocol data unit

RSA – Rivest–Shamir–Adleman

RTS – Real-time strategy

RTT – Round Trip Time

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

ВСТУП

За останні роки ігрова індустрія стала займати значну частину у сфері розваг. І наступні років 10 темп розвитку відеоігор не зміниться. Відеогра – це електронна гра, в ігровому процесі якої гравець використовує інтерфейс користувача, щоб отримати зворотною інформацію з відеопристрою.

Тому необхідно постійно вдосконалювати технології створення відеоігор, починаючи від оптимізації на низькому рівні (безпосередня робота з комп'ютерною апаратурою і мережами та інше) до вдосконалення існуючих архітектур, шаблонів проектування, алгоритмів на програмному рівні.

Метою дипломного проекту є створення архітектури для онлайн-гри з самого початку на основі існуючих рівень і оптимізації використаних алгоритмів.

Практична цінність отриманих результатів. В результаті виконаного аналізу літератури розроблена архітектура і компоненти програмного забезпечення, що реалізують онлайн-ігри клієнт-серверної архітектури. Використані алгоритми є загальноприйнятими для розробки даного типу ігор, але в ході виконання проєкту були модифіковані саме під визначену функціональність гри.

Практична цінність дипломної роботи полягає у структуризації використаної інформації та її практичному застосуванні у сфері розробки онлайн-ігор. Також у грі розроблено основу архітектури для синхронізації. А значить проєкт можна використовувати як основу для додавання нових геймлейних механік з використанням розробленого інтерфейсу. Тобто не потрібно досконало вивчати як працює кожен блок архітектури, щоб його використовувати. Достатньо знати, який буде результат виконання певної функції із заданими параметрами.

Метою даної роботи є розробка структури онлайн-гри клієнт-серверної архітектури та програмного забезпечення для неї на основі теоретичних відомостей про особливості розробки онлайн-гри.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ РОЗРОБКИ ОНЛАЙН-ГРИ

Як показує практика, гравцям необхідно отримувати таку ж якість гри, як і в офлайн-іграх: миттєва реакція на дію гравця без жодних затримок, ідентичний стан гри для всіх, плавний рух об’єктів. Але оскільки онлайн-ігри використовують мережу Інтернет, виникає ряд проблем, що погіршують якість гри.

1.1 Object Replication

Проблема реплікації є першою проблемою, яку необхідно вирішити, щоб забезпечити гравцям комфортну гру. Мета реплікації – забезпечити, щоб усі гравці в грі мали однакову картинку.

Проблема реплікації вперше була вивчена в розподіленій обчислювальній системі як засіб для підвищення відмовостійкості системи та підвищення її продуктивності. У цьому сенсі відеоігри є нетиповою розподіленою системою, у якій реплікація є необхідністю, а не просто засобом для досягнення мети.

Рішення проблеми реплікації зазвичай класифікуються на дві основні категорії, а при застосування до відеоігор можна інтерпретувати наступним чином [1]:

- Активна реплікація

  Кожен ввід гравця (наприклад, натиснення клавіш для переміщення) можуть оброблятися незалежно від інших на кожному клієнті, а результат обробки даних (нова позиція об’єкту) передається іншим клієнтам
- Пасивна реплікація

  Дані гравці надсилаються на одну машину(сервер), де відбувається обчислення стану гри на основі отриманих даних і результат обчислень надсилається всім гравцям.

Хоч активна реплікація є очевидним і простим рішенням, не так легко підтримувати її працездатність. Одним із головних недоліків активної

реплікації є те, що вона дуже нестійка. Це означає, що всі гравці повинні бути ініціалізовані однаковою копією стану та підтримувати повне уявлення про неї постійно і для всіх клієнтів (а це спричиняє масовий витік інформації). Оновлення стану та події повинно бути ідеально детермінованим та реалізованим однаково для всіх клієнтів. Навіть найменші відмінності в оновленні стану об’єктів посилюються, що призводить до катастрофічних помилок десинхронізації.

Помилки десинхронізації часто бувають неочевидними. Наприклад, різні компілятори можуть використовувати різні стратегії округлення плаваючої точки, що призводить до розбіжності в обчисленнях нової позиції. Відновитися після синхронізації досить важко і одним з найпростіших рішень є завершення гри.

Пасивна реплікація на відміну від активної реплікації, що намагається підтримувати одночасне моделювання на всіх машинах мережі, в пасивній реплікації є один комп’ютер (сервер), який відповідає за весь стан гри. Гравці надсилають на сервер факт того, що якась клавіша була натиснута. А сервер вже обчислює новий стан гри на основі отриманих даних і надсилає оновлені дані всім клієнтам (гравцям).

Основною перевагою використання пасивної реплікації є її надійність. Також пасивна реплікація дозволяє реалізувати більш надійні заходи проти обману у грі.

1.2 Latency

Слово затримка має різні значення у різних ситуаціях. У контексті комп’ютерний ігор, це означає інтервал часу між спостережуваною причиною та її спостережуваною дією. Залежно від типу гри, це може бути що завгодно. Наприклад, час між клацанням миші і дією у грі (наприклад, вистріл). Затримка неминуча, і різні жанри ігор мають різну затримку і різний поріг прийнятності.

Наприклад, шутери є чутливими до затримки. Затримка 16 – 50 мс є припустимою у шутерах, і гра буде комфортною для гравця. Затримка 50 – 150 мс вже буде більш відчутною, але грати цілком комфортно. Якщо затримка більше 150 мс, то незалежно від частоти кадрів, гравець буде відчувати, що гра стала більш млявою і неохоче відповідає на дії гравця. Ігри RTS є найбільш толерантними до затримки. Затримка в цих іграх може досягати 500 мс, не завдаючи шкоди користувачеві.

Хоча існує багато джерел затримки, затримка, яку зазнає пакет, коли він подорожує від джерела до місця призначення, є найбільш важливою причиною затримки у мультиплеєрі. Існує чотири основні затримки роботи пакету протягом його життя.

1. Затримка обробки.

   Мережевий маршрутизатор працює, отримуючи пакети з мережевого інтерфейсу, вивчаючи цільову IP-адресу, з'ясовуючи наступну машину, яка повинна отримати пакет, а потім пересилає його з відповідного інтерфейсу. Час, що був витрачений на вивчення адреси джерела і визначення відповідного маршруту і є затримкою обробки. Затримка обробки також може включати будь-яку додаткову функціональність, яку надає маршрутизатор, наприклад, NAT або шифрування.

2. Затримка передачі.

   Щоб маршрутизатор пересилав пакет, він повинен мати інтерфейс зв'язку, що дозволяє йому пересилати пакет уздовж фізичного середовища. Наприклад, 1 MB Інтернет дозволяє передавати приблизно 1 мільйон біт. Таким чином, потрібно близько 12,5 мс, щоб передати 1500-байтний пакет. Цей час витрачений на передачу і є затримкою передачі.

3. Затримка черги.

   Роутер може обробляти лише обмежену кількість пакетів одночасно. Якщо пакети приходять швидше ніж роутер їх обробляє, то нові пакети заносяться в чергу прийому.

4. Затримка розповсюдження.

Здебільшого, незалежно від середовища, інформація не може передаватися швидше ніж швидкість світла. Таким чином, навіть якби інформація передавалися зі швидкістю світла, все одно на великі відстані буде існувати затримка декілька мс.

Деякі з наведених затримок можна оптимізувати, а деякі ні. Затримка обробки зазвичай є незначним фактором. Затримка передачі зазвичай залежить від типу з'єднання. Затримка черги – це результат резервного копіювання пакетів, що очікують на передачу чи обробку.

Мінімізація затримки обробки та передачі допоможе мінімізацію затримку черги. Варто зазначити, що оскільки типова маршрутизація вимагає вивчення лише заголовка пакету, можна зменшити затримку черги для всіх пакетів, надіславши кілька великих пакетів замість багатьох невеликих пакетів.

Затримка розповсюдження часто є хорошою ціллю для оптимізації. Тому що вона заснована на довжині дроту між хостами, що обмінюються даними. Найкращим способом це оптимізувати є переміщення гравців ближче один до одного. У P2P топології це означає пріоритетність географічного положення при пошуку нових гравців у ігрову сесію. У CS топології це означає, що ігрові сервіси є близькими до гравців. Іноді фізичної місцевості недостатньо для забезпечення затримки з низьким розповсюдженням: прямі з’єднання можуть не існувати, а це вимагає наявності маршрутизаторів для маршрутизації трафіку.

1.3 Надійність передачі пакетів

Оскільки необхідно забезпечувати певний рівень надійності в мережі, майже в кожній грі для багатьох гравців, важливим рішенням, яке слід прийняти на початку розробки, є рішення між протоколами TCP та UDP [1].

Основна перевага TCP полягає в тому, що він забезпечує стабільну реалізацію надійності. Без додаткових зусиль гарантує, що всі дані будуть доставлені в порядку, в якому пакети були відправлені.

Основним недоліком TCP є те, що все, що він надсилає, повинно надійно відправлятися та оброблятися по порядку. У багатокористувацькій грі зі швидкою зміною станів гри існують три сценарії, де особливості TCP спричинятимуть наступні проблеми.

1. Втрата даних з низьким пріоритетом, що заважають прийому даних з високим пріоритетом. Розглянемо короткий обмін даними між двома гравцями шутера від першої особи клієнт-серверної архітектури. Гравець А на Клієнті А та гравець В на клієнті В стикаються один з одним. Раптом ракета випущена невідомо ким вибухає на відстані, і сервер надсилає пакет клієнту А для відтворення звуку вибуху. Незабаром після цього Гравець В стрибає перед гравцем А і вистрілює, і сервер відправляє пакет з інформацією про гравця В клієнту А. Через неідеальний мережевий трафік, перший пакет з інформацією про вибух ракети опускається, але другий пакет, що містить інформацію про рух гравця В, ні. Звук вибуху має низький пріоритет для гравця А, тоді як ворог, що стріляє в нього, має високий пріоритет. Для гравця А не буде проблемою, якщо звук вибуху не відтвориться, тобто втрата пакету з цією інформацією є не критичною. Однак, оскільки TCP обробляє пакети по порядку, модуль TCP не передасть пакет переміщення гравця в гру при його отриманні. Натомість він буде чекати, поки сервер повторно не відправить пакет, який було втрачено. А це спричиняє додаткову затримку, що вже стає проблемою для гравця А.
2. Два окремих потоки надійних впорядкованих даних, що заважають один одному. Навіть у грі без даних з низьким пріоритетом, в якій всі дані повинні надійно передаватися, система TCP все ще може викликати проблеми. Розглянемо попередній сценарій, але замість вибуху ракети пакет містить повідомлення в чаті. Повідомлення чату можуть мати

критичне значення, тому їх слід надсилати таким чином, що гарантує їх отримання. Крім того, повідомлення в чаті потрібно обробляти по порядку, оскільки невпорядковані повідомлення можуть бути досить заплутаними. Однак повідомлення в чаті потрібно обробляти лише в порядку, що стосується інших повідомлень чату. Гравець А буде незадоволений ситуацією, що втрата пакета повідомлення, перешкоджатиме обробці пакета з переміщенням і вистрілом гравця В.

3. Ретрансляція неактуального стану гри. Уявіть, що гравець В пробігає всю карту, щоб вистрілити в гравця А. Він починає з позиції x = 0 і протягом 5 секунд переміщається до позиції x = 100. Сервер надсилає пакети гравцеві А п’ять разів на секунду. У кожному пакеті знаходиться координата останньої позиції гравця В. Якщо сервер виявить, що будь-який або всі пакети не дійшли, він повторно їх відправить. Це означає, що в той час, коли гравець В наближається до кінцевої позиції x = 100, сервер може повторно передати старі дані про стан гравця, в яких координати гравця В близькі до початкової позиції. Це призводить до того, що гравець А бачить застарілі дані про положення гравця В, тобто не бачить, що він знаходить поруч і стріляє в нього. А це неприйнятний досвід для гравця А.

Окрім описаної вище проблеми, є ще кілька недоліків використання TCP. Хоч контроль перевантаженості дозволяє запобігти втраченим пакетам, він не може бути рівномірно налаштований для всіх платформ. Алгоритм Nagle є причиною проблеми затримки відправки пакетів. Насправді ігри, які використовують TCP, як правило відключають алгоритм Nagle, щоб уникнути цієї проблеми, хоч це і спричиняє збільшення кількості втрачених пакетів.

Також TCP виділяє багато ресурсів для управління з’єднання та відстеження всіх даних, які можливо прийдеться відправити повторно.

UDP, з іншого боку, не пропонує жодної вбудованої надійності та контролю потоку, які забезпечує TCP. Однак він представляє основу, на яку можна надбудувати будь-яку надійність, яку вимагає конкретна гра.

У більшості випадків вибір того, який транспортний протокол використовувати, зводиться до питання: чи потрібно отримувати кожну частинку даних, яку гра надсилає, і чи потрібно її обробляти впорядковано? Якщо відповідь так, то слід використовувати TCP. Якщо TCP не є ідеальним для конкретної гри, то використовують UDP. Або використовують власний так званий RUDP (UDP з надбудовою для забезпечення необхідного рівня надійності). Нижче наведена таблиця особливостей TCP/UDP протоколів (таблиця 1).

Таблиця 1 - Порівняння TPC і UDP сегментів

	TCP	UDP
Надійність	Вбудована. Все доставлено і обробляється в тому порядку, в якому було відправлено	Жодної. Необхідна спеціальна реалізація, але це не забезпечує ідеальну надійність
Контроль потоку	Автоматично уповільнить швидкість передачі, щоб уникнути втрату пакетів	За бажанням розробляють власний контролер перевантаженості потоку
Вимоги до пам'яті	OS повинна зберігати копії всіх відправлених даних до моменту підтвердження їх отримання	Немає. Необхідна власна реалізація, яка визначатиме, які дані необхідно зберігати, а які ні

Висновки до розділу 1

У даному розділі було розглянуто основні проблеми, які необхідно вирішити при створенні онлайн-гри.

З проведено аналізу можна зробити висновок, що кожну з описаних проблем вирішують загальними методами з удосконаленнями, які підходять конкретному жанру гри з конкретними ігровими механіками.

Показано, що більшість ігрових студій мають власні розробки для вирішення проблем, які вони використовують як онову для своїх проектів додаючи невеликі модифікації. Інді-студії, зазвичай, використовують відкриті бібліотеки і модифікують їх для власних потреб.

2. АНАЛІЗ МЕРЕЖЕВИХ АРХІТЕКТУР

Проаналізуємо мережеві архітектури, через які відбувається взаємодія між гравцями. На сьогодні є дві основні архітектури.

2.1 Аналіз архітектури Peer-To-Peer

Peer-To-Peer(P2P) архітектура – варіант архітектури системи, в основі якої стоїть ідея рівноправності вузлів. Рисунок 1 ілюструє дану топологію [2].

У топології P2P кожен окремий учасник з'єднаний з кожним іншим учасником. Це означає, що між клієнтами передається велика кількість даних. Кількість з'єднань – це квадратична функція, тобто якщо задано n вузлів, то кожен вузол повинен мати n-1 з'єднань. В результаті вимоги до пропускної спроможності для кожного вузла збільшується, коли збільшується кількість підключених до мережі вузлів. Однак, на відміну від клієнт-серверної архітектури, вимоги до пропускної спроможності однакові для всіх вузлів.

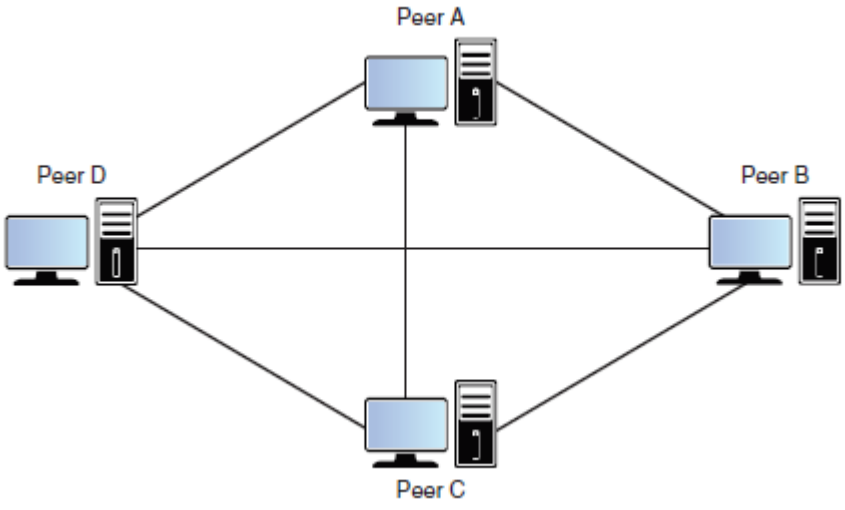

Рисунок 1 - Топологія Peer-To-Peer

Поняття авторитету більш розмите, ніж у клієнт-серверній архітектурі. Один з можливих підходів полягає в тому, щоб надати більші права одному з вузлів над певними частинами гри. Наприклад, управлінням AI може займатися

лише один з вузлів, а іншим передавати лише результат обчислень (поворот, позиція).

Також визначення авторитету вирішує ще одну проблему P2P архітектури, а саме підключення нових гравців. Зазвичай, для зручності, гравець вводить IP-адресу авторитетного гравця, який ініціалізує ігрову сесію. У відповідь на підключення новий гравець отримує список вже підключених раніше гравців. І після, можна підключитися до кожного вузла.

Недоліком даної архітектури є відсутність безпеки від шахрайства. Розглянемо приклад: два гравця грають в покрокову гру, де кожен з них повинен атакувати іншого під час свого ходу. Нехай перший гравець буде хакером. Перший гравець, під час свого ходу повинен нанести певну шкоду (damage), який потім відправляється іншому гравцю. Але оскільки обчислення шкоди (damage) обраховується локально на машині хакера, то ці дані можна легко замінити. І в результаті, другий гравець отримує 9999 шкоди і програє.

Один з варіантів рішення даної проблеми є додати перевірку вхідних даних на коректність. Наприклад, якщо шкода (damage) не входить в певний діапазон, то ігноруємо значення, яке перевіряємо. Але в даному рішенні хакер може налаштувати фільтр таким чином, що тепер просто відкидає всі атаки другого гравця. Саме з таких ситуацій можна зробити висновок, що жоден з вузлів не повинен відповідати за дані гри, інакше не можна бути впевненим в тому, що дані вірні.

Також варто згадати й іншу серйозну проблему P2P архітектури. А саме спірні ситуації, які виникають через затримку пакетів. Наприклад, у грі, де гравці ходять одночасно і переміщаються по клітинкам, але в одній клітинці може бути лише один гравець. Саме через затримку можлива ситуація, коли два гравці переміщаються у одну клітинку, оскільки ще не знають, що вона вже зайнята.

2.2 Аналіз архітектури Client-Server

У топології клієнт-сервер один ігровий екземпляр позначається сервером, а всі інші позначаються як клієнти. Кожен клієнт спілкується лише з сервером і про інших клієнтів може і не знати [3]. Сервер відповідає за спілкування з усіма клієнтами. Рисунок 2 ілюструє дану топологію.

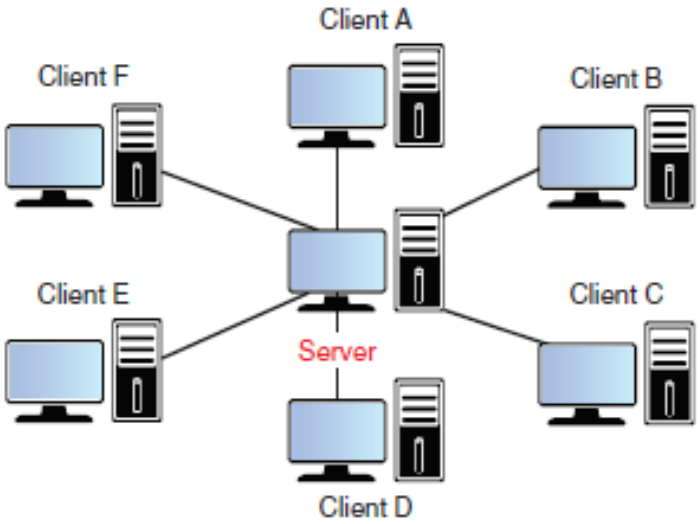

Рисунок 2 - Топологія Clent-Server

У топології з заданими n клієнтами існує загальна кількість 2n з'єднань. Однак розподіл з'єднань несиметричний, оскільки сервер матиме n з'єднань (по одному до кожного клієнта), а кожен клієнт матиме лише одне підключення до сервера. З точки зору пропускної спроможності, якщо є n клієнтів і кожен клієнт відправляє b байтів на секунду даних, то сервер повинен мати достатню пропускну спроможність, щоб обробляти $\boldsymbol{b} * \boldsymbol{n}$ вхідних байтів в секунду. Аналогічно, якщо серверу необхідно відправити *c* байтів в секунду для кожного клієнта, сервер повинен підтримувати $\boldsymbol{c} * \boldsymbol{n}$ вихідних байтів в секунду. Однак кожному клієнту необхідно підтримувати *c* вхідних байтів в секунду та *b* вихідних байтів в секунду. Це означає, що при збільшені кількості клієнтів пропускна спроможність сервера повинна збільшуватися лінійно. Теоретично,

вимоги пропускної спроможності для клієнта не залежать від кількості клієнтів. Однак на практиці підтримка більшої кількості клієнтів призводить до збільшення кількості реплікацій, а значить і до збільшення кількості пакетів, необхідних для синхронізації гри.

Більшість ігор, реалізованих на архітектурі CS, використовують авторитетний сервер. Це означає, що моделювання ігрового сервера вважається єдиним правильним. Якщо змодельований стан гри на клієнті відрізняється від стану гри на сервері, то клієнт повинен оновити свій стан на основі того, що скаже сервер.

Оскільки доступ до даних гри має лише сервер, всі обчислення відбувають на сервері, а клієнтам передається вже результат обчислень. Оскільки клієнти не можуть маніпулювати даними гри це унеможливлює деякі варіанти шахрайства. Так у грі, наведеній у попередньому підрозділі, хакер не зможе підмінити дані про шкоду (damage) у пакеті, оскільки архітектура CS передбачає відправлення від клієнта пакет з інформацією, що натиснута клавіша нанесення шкоди (damage).

Існує підкласифікація типів серверів, а саме виділені сервери. Виділені сервери управляють лише обчисленням ігрового стану і, як правило, не відображають жодної графіки. Цей тип сервера часто використовується великобюджетними іграми.

Висновки до розділу 2

У даному розділі було розглянуто та проаналізовано дві основні архітектури взаємодії вузлів мережі.

З проведеного аналізу зроблено висновок, що вибір топології є одним із найважливіших рішень, які приймаються при створенні мережевої гри. Топологія мережі визначає, як комп'ютери в мережі підключаються один до одного. У контексті гри топологія визначає, як будуть організовані комп'ютери, що беруть участь у грі, щоб усі гравці могли бачити актуальну версію стану гри. Як і у випадку з рішенням мережевого протоколу, існують компроміси незалежно від обраної топології.

Показано, що вибір архітектури залежить від жанру гри, ігрової механіки та масштабності гри. Наприклад, для невеликої інді-гри, що розрахована на локальне підключення гравців, можна обрати архітектуру P2P. Для масштабних проєктів необхідно реалізувати архітектуру CS для забезпечення безпеки і надійності ігрового процесу, але для її реалізації необхідно мати окремий потужний комп’ютер, що виконував би роль сервера.

3. РОЗРОБКА ОНЛАЙН-ГРИ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

3.1 Архітектура програми

На рисунку 3 представлено архітектуру програми, що відповідає за синхронізацію об’єктів.

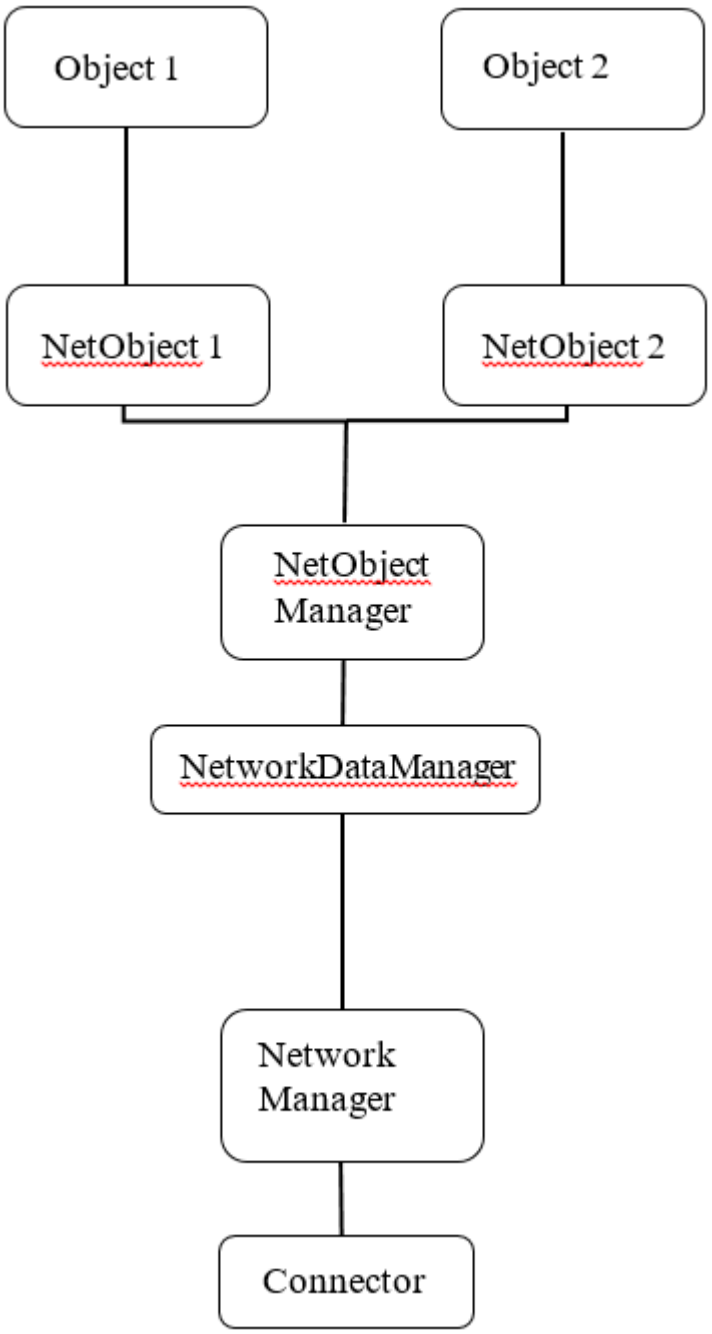

Рисунок 3 - Архітектура програми

Object – об’єкт, який необхідно синхронізувати. Має функцію, що обробляє дані отримані з мережі. Тобто кожен об’єкт знає, які дані необхідно відправити і необхідно отримати для синхронізації. Якщо об’єкт отримав повідомлення, яке він не знає як обробити, то повідомлення ігнорується.

NetObject – клас для синхронізації об’єктів.

Функції класу:

- CreateMessage() – отримує дані з об’єкту і упаковує їх у екземпляр класу Message. Далі отриманий екземпляр відправляється у NetObjectManager.
- Update() – викликається функція об’єкту, що обробляє дані отримані з мережі.
- SetMessageHandler() – функція за допомогою якої встановлюємо функцію обробки повідомлення.

Члени (members) класу:

- m_MessageHandler – вказівник на функцію обробки повідомлень.
- m_IsMaster – булева змінна, що визначає чи є даний об’єкт мастером чи реплікою.
- m_Messages – вектор повідомлень, які необхідно обробити.
- m_Index – індекс NetObject. Завдяки цьому індексу програма може синхронізувати NetObjects на різних PC.
- NetObjectManager – клас для управління NetObjects.

Функції класу:

- Init() – ініціалізація вектору вказівників на NetObject.
- Register() – реєстрація нового NetObject для об’єкту, який необхідно синхронізувати.
- Delete() – функція для видалення елементів з вектору.
- Update() – фукнція що викликає функцію Update() кожного зареєстрованого NetObject.

- TakeMessage() – фукнція для отримання повідомлення з NetworkDataManager
- FindTaker() – за параметром, вказаним у класі NetMessage, шукаємо отримувача і додаємо повідомлення до внутрішнього масиву NetObject.

Члени (members) класу:

- m_NetObjects – вектор у якому зберігають вказівники на NetObject.

NetworkDataManager – клас у якому зберігаються повідомлення, які необхідно відправити у мережу і повідомлення, отримані з мережі, які необхідно розподілити між NetObjects для синхронізації.

Функції класу:

- AddMessageFromNetwork() – додаємо повідомлення у вектор m_MessageIntoNetwork.
- SendMessageIntoNetwork() – додаємо повідомлення у вектор m_MessageIntoNetwork.
- GetMessageFromDeque() – повертає останнє повідомлення з вектору m_MessageFromNetwork.
- GetMessageForSendInNetwork() – повертає останнє повідомлення з вектору m_MessageIntoNetwork.

Члени (members) класу:

- m_MessageIntoNetwork – вектор, що містить повідомлення, які необхідно відправити.
- m_MessageFromNetwork – вектор, що містить повідомлення, які отримали з мережі і необхідно розподілити між NetObject.

NetworkManager – клас, що керує класом Connector. Це необхідно для того, щоб забезпечити абстрактність архітектури і не взаємодіяти з Connector напряму з інших класів.

Функції класу:

- SetConnector() – встановлює тип коннектору.
- ConnectToHost() – підключення до заданої IP-адреси.
- CreateServer() – ініціалізація коннектора як сервер.
- CreateClient() – ініціалізація коннектора як клаєнта.
- ShutDown() –
- GetMessage() – отримання повідомлення.
- ConnectNewPeers() – підключеннння нового піра.
- SendMessage() – відправлення повідомлення.

Члени (members) класу:

- m_Connector – вказівник на коннектор, який використовується для прийому/відправлення повідомлень.

Connector – клас, що є обгорткою над бібліотекою ENet [4], яка використовується для відправки/прийому пакетів через мережу.

Функції класу:

- Init() – ініціалізація об’єкту.
- ConnectToHost() – приєднання до хоста.
- GetMessage() – отримання повідомлення з мережі.
- ShutDown() –
- SendMessages() – відправка повідомлень.
- NetMessage – допоміжний клас, що містить дані для синхронізації, а також інформацію про отримувача.

Члени (members) класу:

- m_SendTo – інформація кому відправити повідомлення (одному з гравців, серверу чи усім).
- m_MessageType – вказано що саме буде синхронізовано (переміщення об’єкту, подія вистрілу, смерть гравця і т.д.).
- m_ReceiverNetObjectID – індекс NetObject, якому буде надіслано повідомлення.

- m_IsReliable – вказано чи є повідомлення надійним, для визначення типу протоколу передачі даних TCP/UDP.
- m_Data – інформація для синхронізації.

Висновки до розділу

У даному розділі було розглянуто сформовану архітектуру. Розглянуто короткий опис класів та ключових функцій, які виконують всю логіку прийому/передачі інформації. Продемонстровано структурну схему розробленої архітектури для забезпечення синхронізації.

Описана архітектура є основою гри, яка забезпечує комфортну гру гравцям. З використанням даної основи було модифіковано ігрові механіки для онлайн-режиму.

4. ОПИС РОЗРОБЛЕНИХ АЛГОРИТМІВ ТА ПІДПРОГРАМ

4.1 Опис роботи архітектури Client-Server у програмі

Для того, щоб почати нову гру, один з гравців повинен обрати мод гри і карту. Тоді цей гравець виступає у ролі сервера, до якого гравці-клієнти приєднуються. Для того щоб приєднатися до гри, гравець повинен ввести IP-адресу і порт сервера (рисунок 4).

Рисунок 4 – Меню вводу для приєднання до сервера

Коли гравець приєднався до сервера, то у відповідь отримує повідомлення з інформацією про обрані сервером мод гри і карту, після чого завантажує їх. Далі від серверу клієнт отримує інформацію про вже приєднаних гравців.

У Lobby меню (рисунок 5) гравці можуть обрати колір танку. Щойно гравець змінив колір, то він відправляє повідомлення з інформацією про вибраний колір серверу. Оскільки у програмі використовується архітектура CS, це зменшує ймовірність виникнення ситуації, коли у кілька гравців обрали однаковий колір.

Рисунок 5 – Lobby menu

Додатково добавлено для гравців параметр isReady, який відмічає, що гравець обрав всі можливі параметри для гри, які повинні синхронізуватися. Даний параметр використовується для уникнення ситуації, коли не всі параметри синхронізовано.

Почати гру може лише сервер і лише коли всі гравці готові до гри.

Під час гри гравець-клієнт надсилає на сервер інформацію про ввід з клавіатури. Сервер обробляє отриману інформацію, а результат (наприклад, оновлена позиція танку) надсилається всім клієнтам. Вся ігрова логіка відбувається на сервері, а всім клієнтам надсилається результат обробки.

Умова завершення гри перевіряється лише на сервері, і коли умова виконується, то сервер надсилає повідомлення всі клієнтам для зміни стану гри.

4.2 Синхронізація подій

Оскільки дана інформація є важливою для гравця і якщо втратити дані, то подальша гра неможлива, необхідно для синхронізації подій використовувати протокол TCP. Це означає, що навіть при втраті пакету інформація буде передана повторно, а це гарантує, що клієнт отримає дані.

Спершу необхідно було забезпечити приєднання клієнта до сервера і отримання інформації про обраний сервером мод гри і карту. Файли з описом модів гри і карт є важливою частиною гри, а значить всі вони гарантовано є на PC разом з виконуючим файлом, і запуск гри без них є неможливим. Під час запуску гри всі файли з картами і модами завантажуються і зберігаються у std::map, де ключ це назва карти, а значення – шлях до файлу з інформацією про карту. Тому для передачі інформації про обрані сервером параметри гри достатньо надіслати лише назви моду гри і карти. Також при підключенні клієнта до сервера першому передається індекс танка, яким гравець буде керувати. Оскільки гра написана на архітектурі CS, завданням про розподіл індексів танків між гравцями займається сервер, що унеможливлює ситуацію, коли двом гравцям було надіслано один індекс.

Для синхронізації кольору у програмі є клас ColorList, що містить список класів Color. У класі Color міститься інформація про номер кольору у форматі ARGB і булеву змінну m_isUsed , що визначає чи використовується даний колір зі списку. Клас ColorList містить екземпляр класу NetObject, який використовується для синхронізації. А синхронізується саме параметр isUsed. Наприклад, за замовчуванням гравцям визначено наступні кольори : синій, червоний, жовтий, зелений (рисунок 6). Це перші 4 кольори, які містяться у визначеному у коді списку кольорів для гравців. Весь список кольорів - це синій, червоний, жовтий, зелений, сірий, голубий, фіолетовий.

А значить, що у списку кольорів екземпляри класу Color, що містять використані за замовчуванням кольори, у всіх змінна m_IsUsed дорівнює false. Далі, якщо клієнт захоче змінити колір на інший, то отримає у результаті сірий колір, оскільки це перший у списку невикористаний колір. Після цього необхідно надіслати пакет з інформацією, що синій колір вже не використовується, а сірий зайнято і пакет з інформацією, що саме нульовий гравець тепер має сірий колір. Проблемою даного методу є можливість виникнення ситуації коли кілька гравців мають однаковий колір. Дана проблема є результатом наявності затримки у передачі пакетів.

Рисунок 6 - Кольори гравців за замовчуванням

Синхронізація готовності гравців. Даний параметр було введено для уникнення ситуації, коли один з гравців ще не обрав параметри свого танку. Синхронізується схожим чином, як і колір. Гравець визначає, що вже готовий і пакет з інформацією про готовність надсилається серверу, а потім сервер надсилає інформацію іншим клієнтам.

Синхронізація зміни станів гри. Гра має наступні стани : Lobby menu (підключення клієнтів до гри), in-game (сама гра, де гравці керують танками) – end-game(гравцям представлена таблиця результату гри), pause. За зміну станів гри повинен відповідати сервер і надсилати інформацію про зміну стану гри всім клієнтам. Окремо треба розглянути стан гри pause. У різних жанрах пауза представлена по різному. Наприклад, у одних, якщо один із гравців поставив гру на паузу, то гра призупиняється і для всіх інших гравців. Для багатьох така стратегія є не правильною. Кращим способом вважається, що коли гравець поставив гру на паузу, то лише він не може управляти своїм танком, але інші гравці далі грають і можуть взаємодіяти з гравцем, що поставив гру на паузу. А значить є можливість вбити неактивного гравця.

Синхронізація статистики гравця. За правилами розробленої гри, коли гравець вбиває іншого гравця, то отримує +1 бал до статистики. Оскільки вся

фізика обробляється на сервері, то і за зарахування балів відповідає сервер, а потім надсилає інформацію всім клієнтам. Смерть синхронізується аналогічно.

Синхронізація вистрілу і куль. Оскільки дана гра є нетермінованою, немає потреби синхронізувати кулю кожен кадр. А значить, коли клієнт натискає кнопку вистрілу, до сервера надсилається запит на вистріл, після чого інформація про кулю (тип, позиція, поворот) надсилається клієнтам. Клієнти, використовуючи отриману інформацію, створюють кулю у своїй грі.

4.3 Синхронізація об’єктів

Розроблена гра має об’єкт танк, яким гравець керує. А тому необхідно кожен кадр надсилати пакети для синхронізації позиції танку, щоб всі гравці спостерігали один стан гри. Найпростішим варіантом є надсилати в кінці фрейму інформацію про позицію і поворот кожного компоненту танку. Недоліки даного варіанту:

- Чутливість до втрати пакетів. Для гравця важливо, щоб танк рухався плавно. Але при збільшені відсотку втрачених пакетів плавність втрачається і грати стає неможливо.
- Необхідно надсилати пакети синхронізації кожен кадр для забезпечення плавності переміщення. А значить, збільшення кількості об’єктів, які необхідно синхронізувати, призведе до перевантаженості мережі.

Тому у програмі для синхронізації переміщення використовується dead reckoning.

Більшість аспектів ігрового моделювання є детермінованим, тому клієнт може імітувати їх просто виконавши копію симуляційного коду сервера. Кулі летять таким же чином і на сервері, і на клієнті. Відскакують від стін і підкоряються таким же законам фізики. Однак гравців імітувати неможливо. Це

вносить перелом у плані екстраполяції. У цьому сценарії найкращим рішенням є зробити передбачення руху гравця, яке потім буде коригуватися при необхідності під час оновлення сервера. У мережевій грі dead reckoning (навігаційна система обчислення шляху) – це процес прогнозування поведінки суб'єкту господарювання, виходячи з припущення, що воно буде все те, що робить зараз. Наприклад, якщо це гравець, що рухається прямо, то ми припускаємо, що він продовжить рухатися у цьому напрямку і через певну кількість кадрів буде у певні позиції. Коли імітований об'єкт контролюється гравцем, dead reckoning запускає те саме моделювання, що і сервер, але з умовою відсутності зміни вводу гравця. Це означає, що крім реплікації пози об'єктів, що керуються гравцем, клієнт повинен копіювати будь-які зміни, що використовуються для моделювання наступних позицій. Сюди входить швидкість, прискорення, стан стрибка і більше, залежно від можливостей гри. Поки інші гравці продовжують робити те саме, що робили, dead reckoning дозволяє грі клієнту точно передбачить поточний стан світу на сервері. Однак, коли гравці вживають несподіваних дій, імітація клієнта розходиться зі справжнім станом і повинна бути виправлена (рисунок 7).

Рисунок 7 - Dead reckoning передбачення

Припустимо, що RTT 100 мс і частота кадрів 60 кадрів в секунду. У 0 мс гравець B знаходиться у позиції (0, 0). Через 50 мс клієнт A отримує інформацію, що гравець B знаходиться у позиції (0, 0) і прямує у додатному напрямку X зі швидкістю 1 одиниця в мс. Оскільки цей стан відстає на ½ RTT,

він імітує продовження руху гравця B з постійною швидкістю протягом 50 мілісекунд. На четвертому кадрі гравець A передбачив, що гравець B повинен бути на позиції рівній (117, 0). Далі гравець B у позиції (134, 0) змінює швидкість на (0, 1). Але гравець A отримає інформацію про зміну швидкості лише через 50 мс, а значить реплікація знаходитиметься на позиції (184, 0). А це означає, що локальне передбачення клієнта A відхилилося від справжнього стану світу. Коли клієнт виявить, що локальне моделювання стало неточним, то одним із трьох наступних способів виправляє ситуацію:

- Миттєве оновлення стану. Просто миттєво оновити до нового стану. Недоліком даного способу є те, що гравець може помітить, що об'єкт стрибає на новий стан, але це краще ніж наявність неточних даних.
- Інтерполяція. Отримуючи новий стан, гра може плавно інтерполювати до нового стану протягом заданої кількості кадрів. Це означає, що при кожній синхронізації необхідно запам'ятати додатково новий стан або дельту [5].
- Регулювання стану другого порядку. Навіть інтерполяція може бути неточною, якщо відбулося раптове збільшення швидкості майже нерухомого об'єкту. Щоб бути більш точним, гра може регулювати другим порядком такі параметри як прискорення. Це може бути математично складним, але забезпечить найменш помітні виправлення [6].

Зазвичай ігри використовують комбінацію цих методів, орієнтуючись на величину розбіжності та специфіці гри. Шутер, зазвичай, використовує інтерполяцію для невеликих розбіжностей і телепорт для більших.

Dead reckoning добре працює для віддалених гравців, оскільки гравець не знає, що саме роблять інші. Враховуючи специфікацію розробленої гри, для синхронізації об'єктів було використано dead reckoning з простою інтерполяцією.

4.4 Синхронізація часу

Мережевий протокол часу (NTP) – мережевий протокол для синхронізації часу [7]. У розробленій програмі необхідно, щоб всі клієнти синхронізували свій час з сервером.

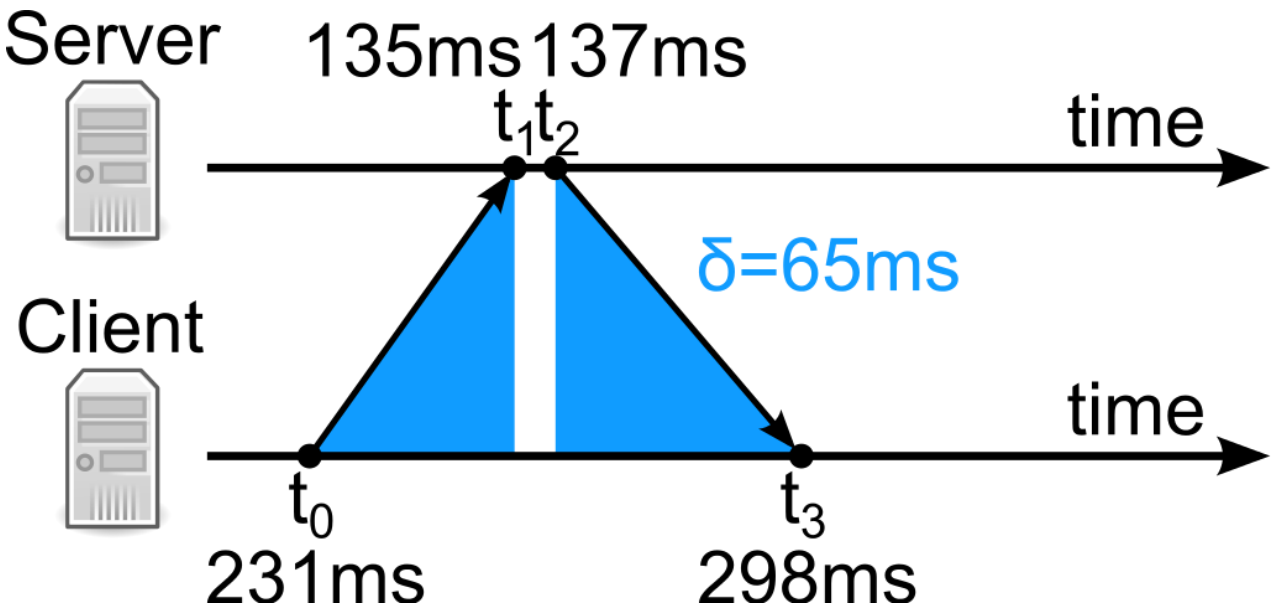

Рисунок 8 - Синхронізація часу

Для синхронізації часу використаємо наступний алгоритм. Типовий клієнт NTP регулярно проводить опитування сервера. Клієнт повинен обчислити своє зміщення часу та затримку в обидва кінці. Зсув часу θ – це різниця в абсолютному часі між двома тактовими годинниками і обчислюється за формулою

$$\boldsymbol{\theta} = \frac{(\boldsymbol{t_1} - \boldsymbol{t_0}) + (\boldsymbol{t_2} - \boldsymbol{t_3})}{\boldsymbol{2}}$$

І затримку в обидва кінця

$$\boldsymbol{\delta} = (\boldsymbol{t_3} - \boldsymbol{t_0}) - (\boldsymbol{t_2} - \boldsymbol{t_1})$$

де:

$\boldsymbol{t_0}$ - мітка часу клієнта передачі пакету запитів,

$\boldsymbol{t_1}$ - мітка часу сервера прийому пакету запитів,

t_2 - мітка часу сервера передачі пакету відповідей,

t_3 - мітка часу клієнта передачі пакету відповідей.

Час затримки і зсув часу необхідно періодично перевіряти для забезпечення точності синхронізації. У розробленій програмі було спроектовано окремий клас NetworkTimeSynchronizer, який кожні 5 хв зі сторони клієнта надсилає запит на синхронізацію часу. Відбувається синхронізація наступним чином.

1. Клієнт надсилає запит на сервер і зберігає час відправки пакету.
2. Коли сервер отримав запит, він запам’ятовує час отримання
3. Сервер надсилає відповідь, у якій містяться дані про час отримання сервером запиту і час відправки відповіді.
4. Клієнт отримує відповідь і запам’ятовує час отримання.
5. За вище наведеними формулами клієнт обчислює час затримки і зсув часу. Обчисленні значення зберігаються у NetworkTimeSynchronizer.

Висновки до заголовку 4

У розділі було розглянуто, як реалізовано алгоритми у проєкті і які проблеми виникали при реалізації.

Розглянуто проблему підключення нових гравців до серверу і синхронізацію інформації для нового гравця було вирішено відправленням додаткових пакетів синхронізації новому гравцю.

Для синхронізації подій у програмі використовується протокол TCP, оскільки необхідно гарантовано надсилати інформації про події для забезпечення одного стану гри для всі клієнтів.

Для синхронізації об'єктів, а саме їх переміщення використовується протокол UDP, оскільки не потрібно гарантовано отримати всі дані про переміщення, основне отримати актуальні дані.

Для зменшення кількості пакетів використовують dead reckoning для передбачення дій гравця. Завдяки цьому у розробленому проекті пакет синхронізації відправляється через кожні 5 кадрів.

Також було реалізовано синхронізацію часу для забезпечення безпеки від шахрайства.

5. БЕЗПЕКА ПЕРЕДАЧІ ІНФОРМАЦІЇ

5.1 Packet Sniffing

При нормальній роботі мережі пакети маршрутизуються через кілька різних комп'ютерів на шляху від джерела до IP-адреси призначення. Як мінімум, маршрутизаторам на шляху потрібно читати інформацію заголовка в пакетах, щоб визначити, куди надіслати пакет. Однак, зважаючи на відкритий характер переданих даних, ніщо не заважає будь-якій машині на маршруті перевіряти всі дані у певному пакеті. Іноді перевірка корисного навантаження, що міститься в пакеті, може бути виконана для забезпечення нормальної роботи мережі. Наприклад, деякі маршрутизатори споживачів застосовують глибоку перевірку пакетів для того, щоб реалізувати якість обслуговування – систему, яка надає пріоритет одним пакетам над іншими.

Але також існує форма перевірки пакетів, яка необов'язково є доброякісною. Packet Sniffing – термін, який зазвичай використовується для опису ситуації, коли читання пакетних даних для цілей, відмінних від звичайної роботи мережі [8]. Це можна здійснити для різних цілей, включаючи спробу крадіжки інформації про вхід або для обману в мережевих іграх.

5.2 Man-in-the-Middle Attack

Під час атаки Man-in-the-Middle Attack комп'ютер, що знаходиться десь на маршруті від джерела до місця призначення, зчитує дані з отриманих пакетів (рисунок 8).

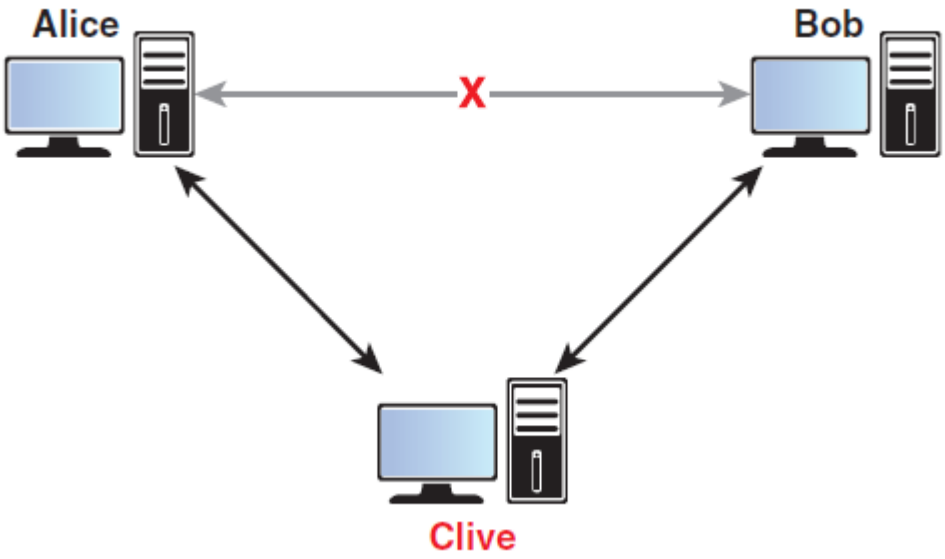

Рисунок 9 - Man-in-the-middle attack, де Clive зчитує інформацію, яку передають один одному Alice і Bob

Існує кілька різних способів, як це може відбутися [9]. Будь-який комп’ютер, що використовує незахищену або загальнодоступну мережу Wi-Fi, може мати всю інформацію про пакет, який було надіслано іншою машиною в спільній мережі. А у проводовій мережі, зчитувати дані з пакету може програма, встановлена на комп'ютері.

Загальний підхід до боротьби з Man-in-the-Middle полягає у шифруванні всіх переданих даних. У випадку з мережевою грою, перш ніж впровадити будь-яку систему шифрування, слід розглянути, чи містить гра будь-які конфіденційні дані, які необхідно зашифрувати.

Якщо у грі є будь-які мікротранзакції, де гравець може придбати товар, необхідно зашифрувати всю інформацію, що пов’язана з покупками. Якщо гра зберігає або обробляє інформацію про кредитку карту, стандарт безпеки даних платіжної картки (Payment Card Industry Data Security Standard) може бути юридичною вимогою. Однак, навіть якщо немає ігрових покупок, будь-яка гра,

в якій гравець заходить в акаунт, який зберігає прогрес, повинен зашифрувати дані, пов’язані з процесом входу.

З іншого боку, якщо єдиними даними, які гра передає по мережі, є дані реплікації, то це не має особливого значення, чи будуть дані перехоплені кимось.

Зокрема, можна використовувати криптографію з відкритим ключем, тип криптографії, що добре підходить для передачі захищеної інформації. Припустимо, Аліса і Боб хочуть передавати зашифровані повідомлення один одному. По-перше, перед тим, як почати спілкуватися між собою, Аліса і Боб генерують різні приватні та відкриті ключі. Приватні ключі залишаються приватними для кожного, хто створив ключ – їм ніколи не слід ділитися з ким-небудь іншим. Коли Аліса і Боб вперше приєднаються один до одного, вони обміняються своїми відкритими ключами. Потім, коли Аліса відправить повідомлення Бобу, вона зашифрує це повідомлення за допомогою відкритого ключа Боба. Потім це повідомлення можна розшифрувати лише за допомогою приватного ключа Боба. По суті, це означає, що Аліса може надсилати повідомлення Бобу, які лише він може прочитати. А Боб може надсилати

повідомлення Алісі, які прочитати зможе лише вона. Це суть криптографії відкритого ключа (рисунок 9).

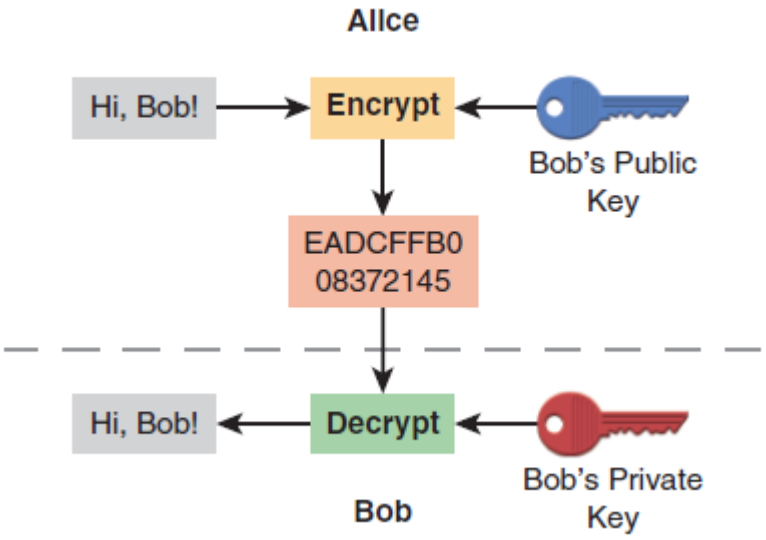

Рисунок 10 - Передача між Алісою і Бобом використовуючи криптографію відкритого ключа

У випадку з мережевою грою, де є сервер для входу, клієнт матиме доступ до відкритого ключа сервера. Коли клієнт бажає ввійти на сервер, їх логін і пароль шифруються за допомогою відкритого ключа сервера. Далі цей пакет може бути розшифрований лише приватним ключем сервера.

Можливо, найпопулярнішою криптографічною системою з відкритим ключем, що застосовується сьогодні, є система RSA [10]. Оскільки RSA – це налагоджена система криптографії, було б марно витрачати ресурси, намагаючись самостійно реалізувати систему. Натомість використаємо надійну реалізацію RSA з відкритим кодом, таку як реалізація передбачена OpenSSL [11].

Є кілька сценаріїв де RSA може бути порушеним. Першим сценарієм було б створення достатньо потужного квантового комп'ютера. Алгоритм Шор (Shor's algorithm) – це квантовий комп'ютерний алгоритм, який може визначати цілі числа в квантовому поліноміальному часі [12].

Інший сценарій полягає в тому, що розроблений алгоритм поліноміального часу для цілочисельної факторизації для стандартних комп’ютерів. Причина, чому це катастрофічно, полягає в тому, що багато надійного спілкування в Інтернеті покладається на RSA або пов’язані з ним алгоритмом.

Якщо RSA зламана, це означає, що багато ключів, які використовуються для HTTPS, SSH та подібних протоколів, більше не будуть захищені.

Висновки до розділу 5

У даному розділі розглянуто проблеми безпеки інформації в онлайн-грі та методи їх вирішення.

Н основі проведеного аналізу у розробленому проекті для реалізації безпечної передачі ігрових даних було використано додатковий алгоритм шифрування даних XOR та бібліотеку OpenSSL, що реалізує систему RSA.

Показано, що масштабні ігрові студії, зазвичай, використовують власні програми для безпечної передачі даних.

6. ОПТИМІЗАЦІЯ РОЗРОБЛЕНОЇ ПРОГРАМИ

Зазвичай перша версія гри реалізує всю заплановану ігрову механіку, але містить багато надлишкових дій і багів. І тому у розробці ігрових проектів є окремий етап оптимізації готового продукту. Під час оптимізації переглядається кожен реалізований модуль гри і оптимізується, якщо це необхідно. Якщо при написанні гри дотримувалися принципу модульності, то оптимізація одного модулю не потребує зміни коду у інших модулях.

6.1 Зменшення кількості відправлених пакетів за один фрейм

У розробленій програмі першим варіантом відправки інформації було створення для кожної події нового повідомлення і зберігання його у масиві до кінця кадру. Після чого всі елементи масиву будуть надіслані кожен окремим пакетом. Але дане рішення є збитковим, оскільки до кожного повідомлення дописуються додаткові заголовки, що збільшують кількість байтів, які передаються мережею. Також є проблема перевантаженості серверу, так як при великій кількості пакетів утворюється черга, а це збільшує час очікування, коли пакет буде оброблено. Для вирішення даної проблеми необхідно розглянути поняття MTU [13].

В комп’ютерних мережах, максимальний блок передачі даних (MTU) – це розмір найбільшого блоку даних протоколу (PDU), який може бути переданий в одній транзакції мережевого рівня.

Стандарт визначає мінімальне значення EtherType, яке становить 1536 байт. Зазначимо, що багато сучасних Ethernet NIC підтримують кадри з MTU вище 1500 байт. Ці jumbo-кадри (jumbo frames) можуть часто мати MTU до 9000 байт.

Значення MTU може бути визначено стандартом, а може бути налаштоване у момент установки з’єднання. Чим вище значення MTU, тим

менше заголовків передається мережею – отож вище і пропускна спроможність. Однак повільні інтерфейси можуть передавати великий пакет значний час, збільшуючи час очікування для інших пакетів. У таблиці 2 наведено основні значення MTU для різних технологій.

Таблиця 2 - Основні значення MTU

Технологія	MTU
DIX Ethernet	1500 байт
Ethernet 802.3	1492 байт
Token Ring (IBM, 16 Мбіт/с)	17 914 байт
Token Ring (802.5, 4 Мбіт/с)	4464 байт
FDDI	4352 байт
X.25	576 байт

У даній розробці будемо використовувати стандарт DIX Ethernet зі значенням MTU 1500 байт.

Максимальний розмір пакету IPv4 становить 65535 байт. А тому виникає питання: якщо пакет, який необхідно передати, більший, ніж значення MTU, то його потрібно фрагментувати. Якщо необхідно передати пакет, розмір якого більший значення MTU, то IP-модуль розіб'є пакет на фрагменти. Фрагменти подібні до звичайних пакетів, але з певними значеннями, встановленими в заголовку. Вони використовують ідентифікацію фрагмента, прапорці фрагментів та поля зміщення фрагмента. Коли IP-модуль розбиває пакет на групу фрагментів, він створює новий IP-пакет для кожного фрагмента і встановлює необхідні поля (рисунок 11). Розглянемо значення полів заголовку фрагменту IP-дейтаграми.

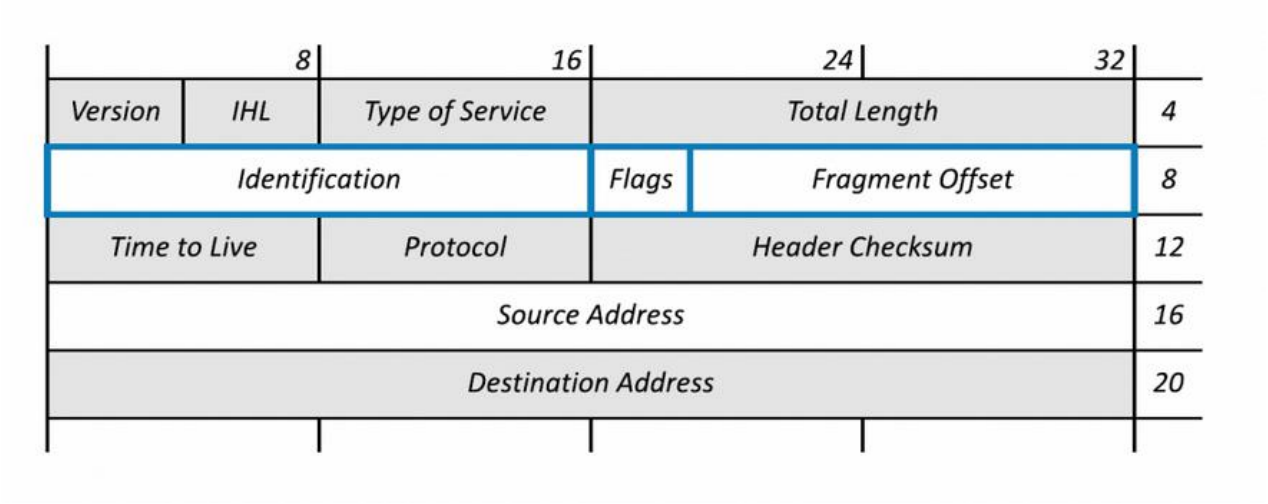

Рисунок 11 - Поля заголовку фрагмента для IPv4.

Поле ідентифікації (The Identification field): 16 біт заповнюється ідентифікаційним номером, унікальним для комбінації адрес джерела, призначення та значення поля протоколу вихідного пакету, що дозволяє розрізняти фрагменти різних пакетів.

Як і в оригінальному пакеті, перший, зарезервований біт поля Прапори (Flags) буде дорівнювати 0 (не встановлено). На відміну від оригінального пакету, всі фрагменти, крім останнього, матимуть третій біт поля, More Fragments (MF) встановлений в 1.

Поле зсув або зміщення фрагменту (Fragment Offset): 13 біт використовується для вказівки початкового положення даних у вихідному пакеті. Ця інформація використовується для повторного збирання даних з усіх фрагментів (які прийшли незалежно від порядку). У першому фрагменті зміщення дорівнює 0, оскільки дані в цьому пакеті починаються там же, що і дані у вихідному пакеті (на початку). У наступних фрагментах значення - це зміщення даних, які фрагмент містить від початку даних у першому фрагменті (зміщення 0), у 8-байтних "блоках" (також октетах). Якщо пакет, що містить 800 байт даних, розділений на два рівні фрагменти, що несуть 400 байт даних, зміщення фрагмента першого фрагмента дорівнює 0, другого фрагмента 50 (400/8). Значення зміщення повинно бути числом 8-байтних блоків даних, що означає, що дані в попередньому фрагменті повинні бути кратними 8-ти байтам.

Останній фрагмент може містити дані, які не кратні 8 байтам, оскільки не буде наступного фрагмента зі зміщенням, який повинен відповідати 8-байтовому правилу.

Поле Загальна довжина (Total Length field): 16 біт змінюється на основі зменшеного розміру даних у фрагменті (плюс IP-заголовок), який дорівнює або менший, ніж MTU. Оскільки поле Зміщення фрагмента у наступних фрагментах повинно бути кратним 8, розмір фрагмента не завжди такий великий, як дозволяє MTU. В IPv4 це поле вказує загальний розмір пакета, включаючи заголовок.

Поле контрольної суми заголовка (Header Checksum field): 16 біт перераховується виходячи зі значення всіх інших полів у заголовку.

У онлайн іграх необхідно уникати фрагментації для уникнення повторного надсилання всього пакету при втраті одного з фрагментів. Саме тому необхідно упаковувати інформацію так, що їх довжина була менше 1400 байт [14].

Але окрім того, що необхідно упаковувати інформацію, необхідно окремо упаковувати інформацію, яка буде передаватися TCP або UPD протоколами.

Тому для вирішення даної проблеми було створено клас Packer, який є компонентом класу NetworkDataManager.

6.2 Винесення в окремі потоки функції прийому та відправки повідомлень

Однією з проблем, яка виникла при розробці гри було перевантаження сервера. Так, наприклад, коли кількість гравців становила більше 5, сервер припиняв рендерити зображення і лише обробляв логіку гри і відправляв повідомлення. Якби сервер був виділений і не міг бути гравцем, описана проблема виникла б дещо пізніше. Але з даним функціоналом гравець, який ініціалізував гру і був сервером, не міг грати, оскільки зображення не

рендерилося. Тому для вирішення необхідно було, окрім зменшення кількості пакетів, винести у різні потоки прийом та відправку повідомлень.

Тому тепер у програмі працюють наступні 5 потоків:

- логіка гри і рендер;
- синхронізація часу;
- прийом повідомлень;
- відправка повідомлень;
- завантаження даних на початку гри.

Після винесення функцій у потоки сервер почав відображати зображення. Але нові потоки на даному етапі виконують свою роботу приблизно за 1/50 кадру. А решту часу даремно викликають функції обробки масивів з вхідними і вихідними пакетами.

Тому необхідно було призупинити потік, поки не почнеться або закінчиться кадр.

Уявимо ситуацію, що необхідно їхати в нічному поїзді. Щоб гарантовано зійти на потрібній станції, прийдеться не спати всю ніч і уважно відслідковувати всі зупинки. Свою станцію не буде пропущено, але буде втома від необхідності спотерігання. Але є і інший варіант: можна глянути у розкладі час прибуття поїзда на необхідну станцію і поставити будильник на потрібний час з невеликим запасом. Цього буде цілком достатньо, але якщо потяг затримується, то ви прокинетеся набагато раніше потрібного часу. Ідеальним рішенням буде покластися на когось, хто розбудить незадовго до станції.

Саме третій варіант і буде застосовуватися у програмі. Для забезпечення такої функціональності у стандартній бібліотеці C++ передбачена умовна змінна. Концептуально вона пов'язана з подією або іншою умовою, і один або кілька потоків можуть сповістить потік, що очікує, що подія настала.

В стандартній бібліотеці C++ представлено дві реалізації умовної змінної: std::condition_variable і std::condition_variable_any. Обидві оголошені в бібліотечному заголовку <condition_variable>. В обох випадках для відповідної синхронізації необхідно працювати з м’ютексами.

Висновки до розділу 6

У даному розділі було доведено та описано, які модулі програми було оптимізовано, чому саме ці модулі потребували оптимізації, як саме їх було оптимізовано.

У розробленому проекті було оптимізовано передачу інформації для зменшення кількості повідомлень і зменшення навантаження на мережу з врахуванням MTU.

Також було оптимізовано механізм передачі та прийому інформації. Обидві функції було винесено в окремі потоки для зменшення навантаження на основний ігровий цикл.

7. ТЕСТУВАННЯ ПРОГРАМИ

Тестування будь-якої програми є важливим етапом розробки. Саме завдяки цьому можна виявити більшість багів і проблем проєкту до його реалізації. Також завдяки тестам можна продемонструвати якість оптимізації проєкту.

7.1 Clumsy

Оскільки гра повинна бути орієнтована на запуск у реальних умовах, необхідно мати можливість тестувати її з різними параметрами мережі. Таким чином, можна визначити слабкі місця програми, її завадостійкість та ефективність dead reckoning.

Саме для такого тестування буде використовуватися програма clumsy [15].

Необхідно розібратися з інтерфейсом програми [16], що зображено на рисунку 12.

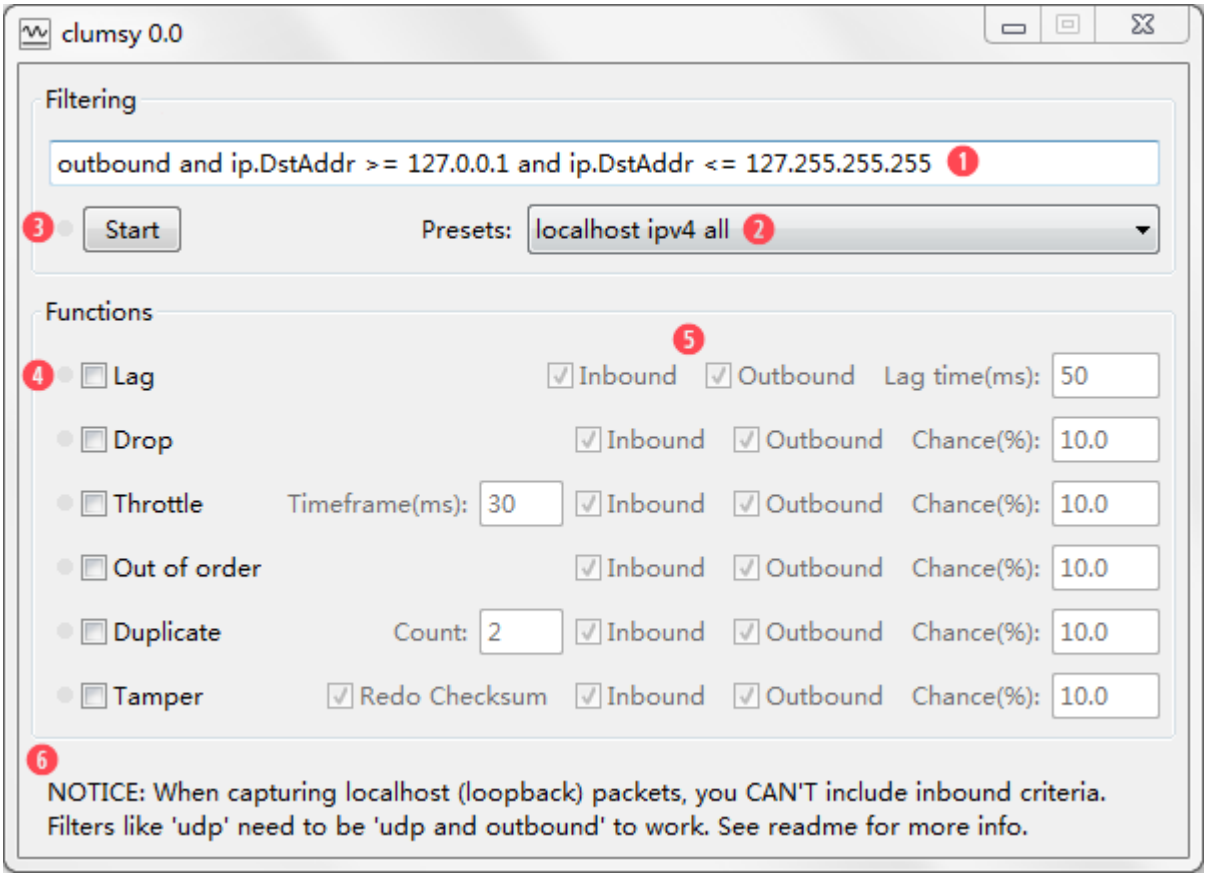

Рисунок 12 - Інтерфейс clumsy

1. Фільтр вводу. Пакети відловлюються на основі цього фільтру.
2. Пресет. Існує список вбудованих пресетів, які можна обрати.
3. Кнопка управління. Натисніть на кнопку, щоб почати процес відловлення пакетів. У деяких випадках, програма може вийти з ладу. Причиною цього може бути неправильний фільтр з точки зору синтаксису. Після запуску текст буде змінено на «Стоп». Зліва невелика іконка, яка стає зеленою, коли програма зловила пакет.
4. Функції контролю. Для застосування функції необхідно встановити прапорець. Можна протягом роботи програми вмикати і вимикати функції без перезапуску програми.
5. Параметри контролю. Для кожної функції є додаткове у правління для налаштування.
6. Статус. Показує кілька корисних текстових повідомлень про поточний стан.

Текст фільтрації безпосередньо передається до WinDivert. Синтаксис тут добре розроблено. Для побудови логічного виразу можна використовувати операції І, НЕ, АБО і круглі дужки. Опції типу =, !=, <, >також присутні. Перелік доступних полів документації WinDivert додається у таблиці 3.

Таблиця 3 - Список доступних полів

outbound	Is outbound?
inbound	Is inbound?
ifIdx	Interface index
subIfIdx	Sub-interface index
ip	Is IPv4?
ipv6	Is IPv6?
icmp	Is ICMP?
icmpv6	Is ICMPv6?
tcp	Is TCP?
udp	Is UDP?
ip.*	IPv4 fields (see DIVERT_IPHDR)
ipv6.*	IPv6 fields (see DIVERT_IPV6HDR)
icmp.*	ICMP fields (see DIVERT_ICMPHDR)
icmpv6.*	ICMPV6 fields (see DIVERT_ICMPV6HDR)
tcp.*	TCP fields (see DIVERT_TCPHDR)
tcp.PayloadLength	The TCP payload length
udp.*	UDP fields (see DIVERT_UDPHDR)
udp.PayloadLength	The UDP payload length

clumsy використовувався для перевірки dead reckoning з різним значенням втрати пакетів, для перевірки надійності передачі TCP пакетів для синхронізації подій з різним значенням втрати пакетів та затримки. І для тестування синхронізації часу з різними параметрами затримки.

За отриманими даними можна зробити наступні висновки.

- При тестуванні dead reckoning при збільшені ймовірності втрати пакетів танк і далі рухається плавно, але чим більше значення втрати пакетів, тим більше танк відхиляється від оригінальної траєкторії. А це створює незручності керування для гравця.
- При тестуванні надійності пакетів для синхронізації подій при збільшені ймовірності втрати пакету збільшується час надходження пакетів. Це пояснюється тим, що при втраті пакету, сервер знов надсилає пакет до тих пір, поки пакет не надійде до клієнта.
- При тестуванні синхронізації часу при збільшенні затримки збільшується різниця у часі між клієнтом і сервером. Але потім через певний час вирівнюється.

Реальними умовами стану мережі для гри вважається наступний стан мережі: 150-200 мс затримки та 5% втрата пакетів.

За даних умов грати комфортно і не помітно жодних помилок синхронізації протягом ігрової сесії, яка триває близько 5-7 хв.

7.2 Tracy Profiler

Tracy Profiler [17] - це наносекундний профіль кадру в реальному часі, який можна використовувати для віддаленої або вбудованої телеметрії програми. Він може профілювати процесор (C, C ++ 11, Lua), GPU (OpenGL, Vulkan) та пам'ять. Він також може відображати блоки, які утримують потоки, та їх взаємодію між собою.

Tracy Profiler розбивається на клієнтську та серверну сторони (рисунок 13). Клієнтська сторона збирає події за допомогою високоефективної черги та

очікує на вхідне з'єднання. Серверна частина підключається до клієнта (рисунок 14) і отримує зібрані дані від клієнта, які потім реконструюються у видиму часову шкалу. Передача здійснюється за допомогою TCP-з'єднання.

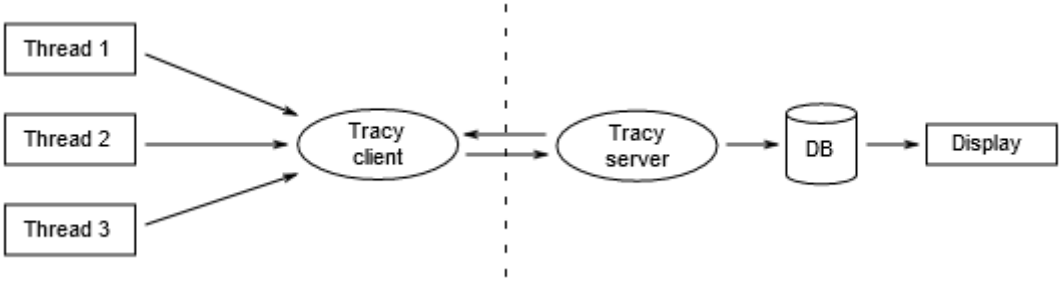

Рисунок 13 - Схема роботи Tracy

Tracy у проекті використовувався для тестування взаємодії потоків.

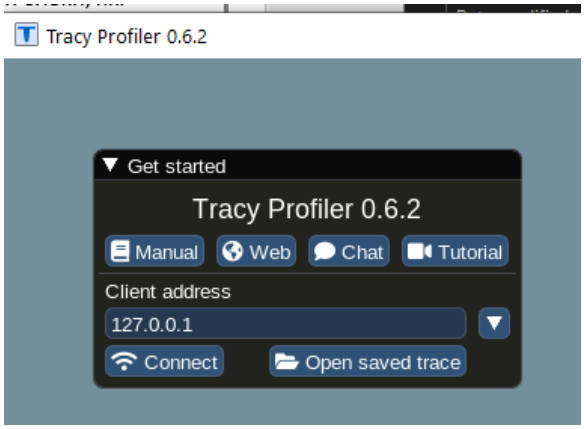

Рисунок 14 - Інтерфейс Tracy для підключення

На рисунку 15 наведено результати тесту програми. Як видно з результатів, у грі працює 5 процесів:

- GameLoop [18] – потік, у якому виконується вся логіка гри і рендер зображення:
- Loading – потік, який працює лише на початку гри для завантаження даних;
- NetworkManager::GetMessage – потік для отримання повідомлень з мережі;

- NetworkManager::SendMessage – потік для надсилання повідомлень у мережу;
- NetObjectManager::TakeMessage – потік для розподілу отриманих з мережі повідомлень між усіма NetObjects.

Рисунок 15 - Результати тестування

На рисунку 16 наведено інформацію про кількість кадрів за секунду. За отриманими даними можна сказати, що гра працює стабільно (60 кадрів за секунду – зелений колір) з просіданням до 58-59 кадрів у деяких моментах (жовтий колір).

Рисунок 16 - Інформація про кількість кадрів за секунду

На рисунку 17 зображено інформацію про завантаженість CPU комп’ютера. Перше збільшення використання CPU відбулося, коли у меню гри було вибрано мод гри і карту. Як видно з рисунку 15, саме в цей момент і почав

працювати потік для завантаження даних. Наступний приріст завантаженості CPU, який відмічено червоною лінією, можна пояснити запуском ще однієї гри на PC.

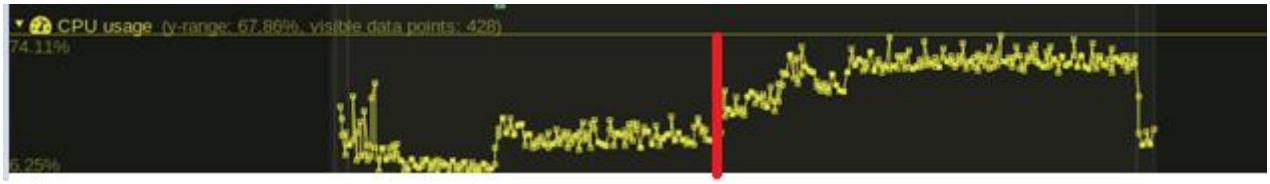

Рисунок 17 - Інформація про використання CPU

1.1 7.3 Аналізатор кількості вхідних і вихідних даних

Для тестування корисної інформації у пакетах і кількості прийнятих/відправлених пакетів потрібно було написати власний аналізатор. Використовуючи даний аналізатор, можна побачити оптимізацію, яка відбувалася внаслідок додавання класу для упакування повідомлень у один пакет. На рисунку 18 зображено інформацію на сервері. У тесті запущено три гри локально на одному PC, де Player 0 – сервер, Player 1, 2 – клієнти.

У аналізаторі відображається інформація про кількість пакетів, яку отримує/надсилає сервер від кожного з’єднання, тобто окремо від кожного клієнта. На рисунку 19 показано дані, отримані з клієнта. Оскільки клієнт підключений лише до сервера, то відображається лише інформація про надіслані/отримані дані з сервера. На рисунку 20, 21 зображено інформацію, зібрану аналізатором, але вже з упакуванням даних.

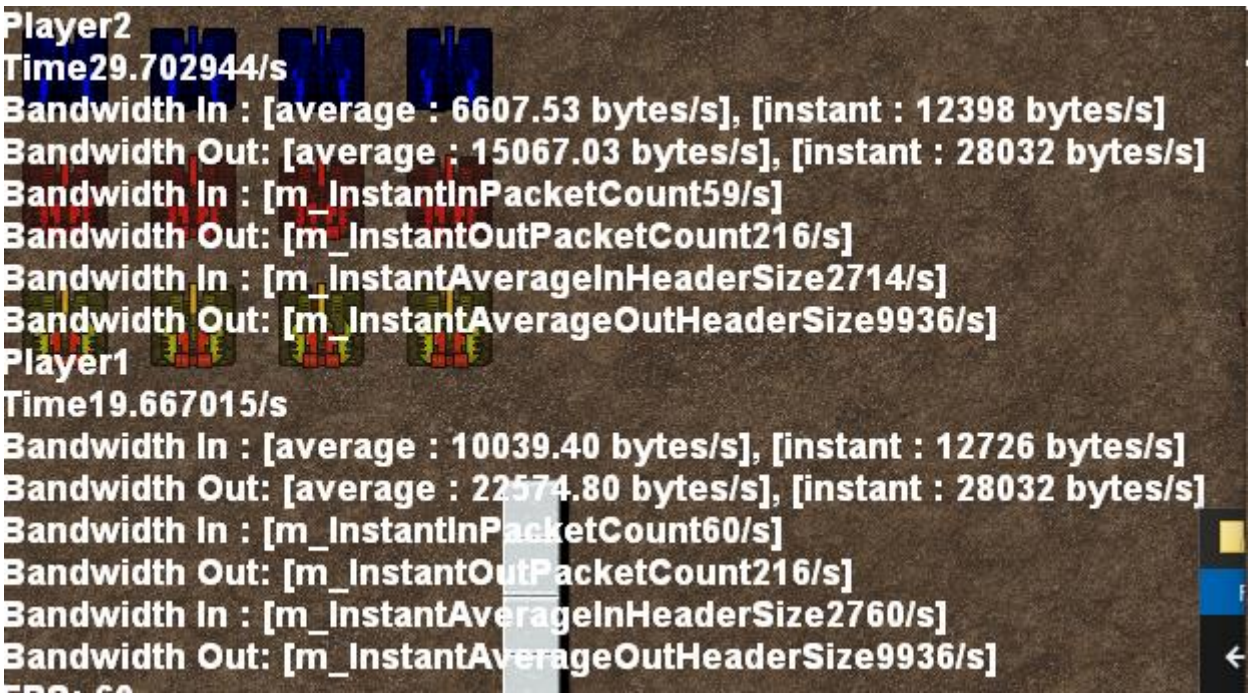

Рисунок 18 - Дані аналізатора на сервері без упакування даних

Рисунок 19 - Дані аналізатора на клієнті без упакування даних

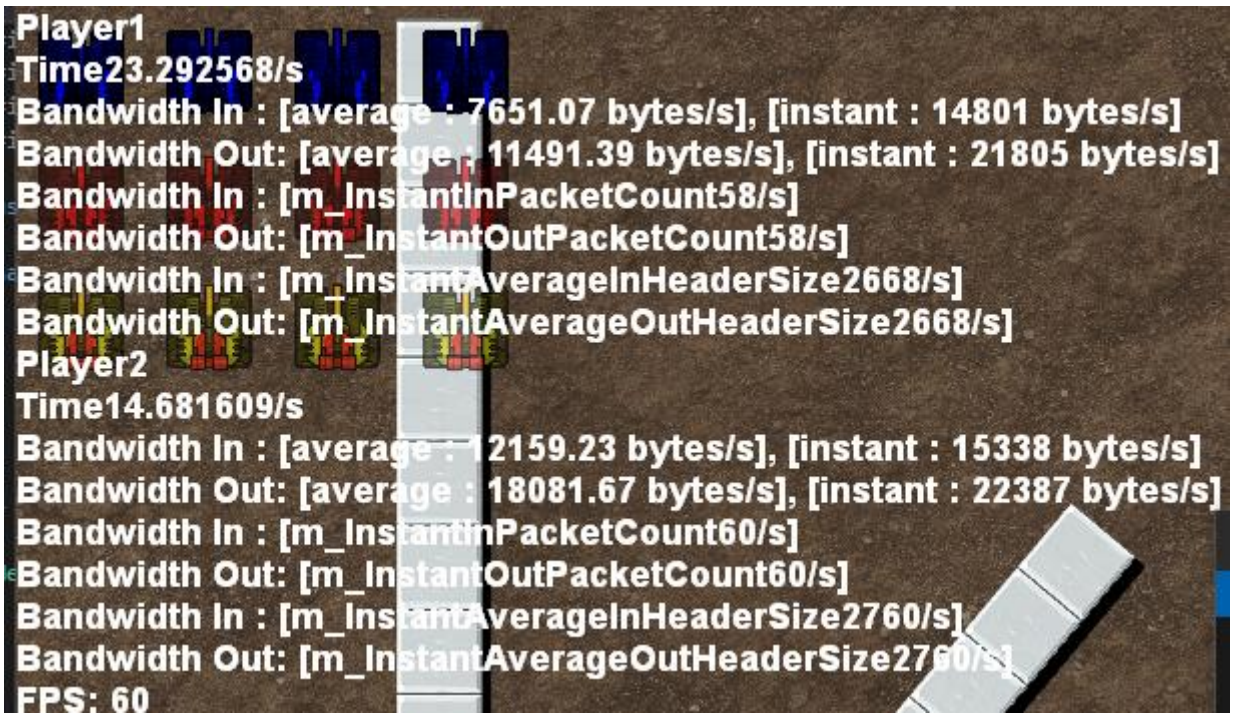

Рисунок 20 - Дані аналізатора на сервері з упакуванням даних

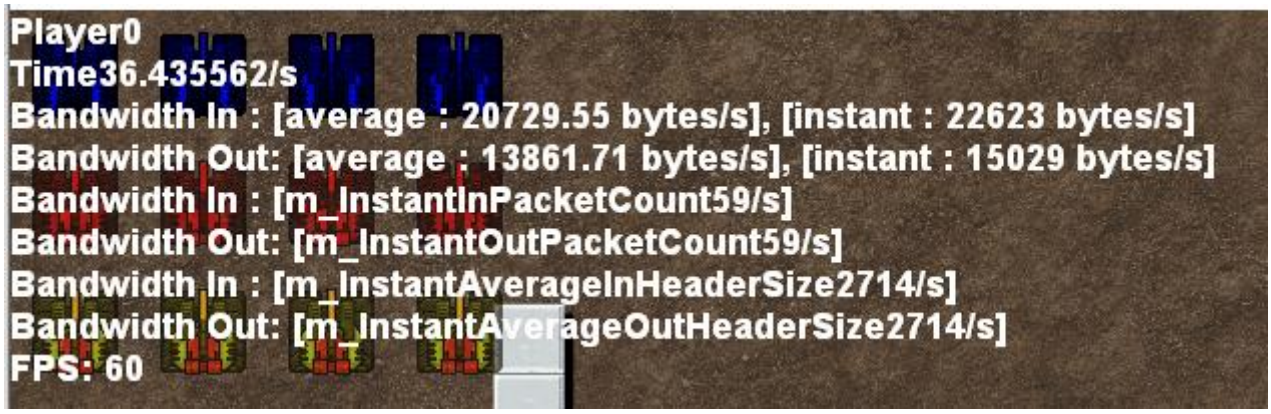

Рисунок 21 - Дані аналізатора на клієнті з упакуванням даних

Порівнюючи дані, отримані з тесту, можна сказати, що завдяки упакуванню даних зменшилася кількість пакетів, яку надсилає сервер клієнтам. А це дозволяє розвантажити мережу. Оскільки зменшилася кількість пакетів, зменшилася і кількість службової інформації, але при цьому необхідно зазначити, що кількість корисної інформації не змінилася.

Висновки до розділу 7

У розділі було розглянуто методи тестування проекту і результати оптимізації деяких процесів. Показано, що використання dead reckoning дозволяє отримати гарний результат при меншій кількості пакетів синхронізації і при досить великому відсотку втрачених пакетів у порівнянні з простим методом реплікації переміщення об’єкту (надсилання кожного кадру позиції об’єкту).

Використання програми Tracy Profiler дозволяє спостерігати за роботою потоків і легко знаходити можливі місця блокування потоків, що полегшує роботу з виправленням ситуації.

Для того, щоб наглядно побачити результат роботи ідеї упакування даних в один пакет, розроблено власний аналізатор. Це дозволило виявити, що цілком можливо запропонований в роботі алгоритм упакування є недостатньо ефективним. Оскільки зменшилася кількість пакетів, що відправляється сервером до клієнтів з 216 до 60, а от кількість пакетів, яку надсилає клієнт серверу, майже не змінилася (з 60 до 58).

Програма clumsy є простою у використанні і існує у вільному доступі. А її функціоналу достатньо, щоб перевірити надійність гри за певних умов мережі.

ЗАГАЛЬНІ ВИСНОВКИ

В роботі розроблено програмне забезпечення онлайн-гри на архітектурі клієнт-сервер. Досліджено різні методи вирішення основних проблем будь-яких мережевих ігор, і обрано найбільш оптимальні з них для розробки програми. Проаналізовано ігровий процес та складності, які виникають при роботі обраних алгоритмів.

Запропоновано варіант реалізації онлайн-гри на архітектурі клієнт-сервер, який не використовується для невеликих проектів через складності, описані в проекті.

Розроблено та описано програми, які необхідні для тестування онлайн-гри і на основі отриманих даних зроблено висновки про ефективність реалізованих алгоритмів.

Розроблено власний варіант геймплею онлайн-гри, що базується на інших іграх з урахуванням побажань користувачів.

Розглянуто можливості використання різних транспортних протоколів для різних цілей з описом їх особливостей.

Розглянуто необхідність шифрування даних і засоби, які забезпечать надійний захист інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Joshua Glazer, Sanjay Madhav Multiplayer game programming : 2016. 386 pages.
2. What are P2P networks and what are they used for? : URL: https://www.digitalcitizen.life/what-is-p2p-peer-to-peer (дата звернення 13.02.2020).
3. Fast-Paced Multiplayer (Part1): Client-Server Game Architecture : URL: https://www.gabrielgambetta.com/client-server-game-architecture.html
4. ENet.h documentation : URL: http://enet.bespin.org/Tutorial.html (дата звернення 19.02.2020).
5. Spline interpolation : URL: https://www.encyclopediaofmath.org/index.php/Spline_interpolation (дата звернення 10.04.2020).
6. Cubic Spline Interpolation : URL: https://timodenk.com/blog/cubic-spline-interpolation/ (дата звернення 10.04.2020).
7. Computer Network Time Synchronization : URL: https://www.eecis.udel.edu/~mills/exec.html (дата звернення 03.04.2020)
8. What is packet sniffing? : URL: https://www.paessler.com/it-explained/packet-sniffing (дата звернення 12.04.2020).
9. What is a man-in-the-middle attack? : URL: https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle-attack.html (дата звернення 12.04.2020).
10. RSA Encryption : URL: https://brilliant.org/wiki/rsa-encryption/ (дата звернення 12.04.2020).
11. OpenSSL : URL: https://www.openssl.org/ (дата звернення 12.04.2020)
12. Elisa Baumer, Jan-Grimo Sobez, Stefan Tessarini Shor's Algorithm: 2015. 26 pages. URL: https://qudev.phys.ethz.ch/static/content/QSIT15/Shors%20Algorithm.pdf (дата звернення 12.04.2020).

13. IP Fragmentation in Detail : URL: https://packetpushers.net/ip-fragmentation-in-detail/ (дата звернення 17.04.2020).
14. Theoretical maximum throughput with Netrounds Test Agents : URL: https://app.netrounds.com/static/2.27/support/defs-notes/theor-thput.html (дата звернення 17.04.2020).
15. Clumsy 0.2 : URL https://jagt.github.io/clumsy/ (дата звернення 07.05.2020)
16. Clumsy manual : URL https://jagt.github.io/clumsy/manual.html (дата звернення 07.05.2020)
17. Tracy Profiler : URL: https://github.com/wolfpld/tracy (дата звернення 10.05.2020)
18. Robert Nystrom Game programming patterns: 2014. 448 pages.